

AMENDMENTS TO THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

- 1 1. (Currently Amended) A method for executing a commit instruction
2 to facilitate transactional execution on a processor, comprising:
3 executing a block of instructions transactionally, wherein executing the
4 block of instructions transactionally involves placing load-marks on cache lines
5 from which data is loaded, placing store-marks on cache lines to which data is
6 stored, and placing transactional stores in a store buffer in the processor during
7 the transaction, wherein the transactional stores are gated and not committed to
8 memory from the store buffer during the transaction; and placing store marks on
9 ~~cache lines to which data is stored;~~
10 encountering the commit instruction during execution of a program,
11 wherein the commit instruction marks the end of a block of instructions to be
12 executed transactionally; and
13 upon encountering the commit instruction, successfully completing
14 transactional execution of the block of instructions preceding the commit
15 instruction, wherein successfully completing the transactional execution involves
16 atomically committing changes made during the transactional execution by:
17 treating store-marked cache lines as locked, thereby causing other
18 processes to wait to access the store-marked cache lines;
19 committing store buffer entries generated during transactional
20 execution to memory, wherein committing each store buffer entry

21 involves removing the store-mark from, and thereby unlocking, a
22 corresponding store-marked cache line;
23 clearing load-marks from cache lines; and
24 committing register file changes made during transactional
25 execution;
26 wherein changes made during the transactional execution are not
27 committed to the architectural state of the processor until the transactional
28 execution successfully completes.

1 2. (Previously Presented) The method of claim 1, wherein
2 successfully completing the transactional execution involves:
3 resuming normal non-transactional execution.

1 3. (Cancelled)

1 4. (Original) The method of claim 1, wherein if an interfering data
2 access from another process is encountered during the transactional execution and
3 prior to encountering the commit instruction, the method further comprises:
4 discarding changes made during the transactional execution; and
5 attempting to re-execute the block of instructions.

1 5. (Previously Presented) The method of claim 1, wherein for a
2 variation of the commit instruction, successfully completing the transactional
3 execution involves:
4 commencing transactional execution of the block of instructions following
5 the commit instruction.

1 6. (Original) The method of claim 1, wherein potentially interfering
2 data accesses from other processes are allowed to proceed during the transactional
3 execution of the block of instructions.

1 7. (Original) The method of claim 1, wherein the block of
2 instructions to be executed transactionally comprises a critical section.

1 8. (Original) The method of claim 1, wherein the commit instruction
2 is a native machine code instruction of the processor.

1 9. (Original) The method of claim 1, wherein the commit instruction
2 is defined in a platform-independent programming language.

1 10. (Currently Amended) A computer system that supports a commit
2 instruction to facilitate transactional execution, wherein the commit instruction
3 marks the end of a block of instructions to be executed transactionally,
4 comprising:
5 a processor;~~and~~
6 a store buffer in the processor; and
7 an execution mechanism within the processor, wherein the execution
8 mechanism is configured to place load-marks on cache lines from which data is
9 loaded, place store-marks on cache lines to which data is stored, and place
10 transactional stores in the store buffer during the transaction, wherein the
11 transactional stores are gated and not committed to memory from the store buffer
12 during the transaction; and place store-marks on cache lines to which data is
13 stored during transactional execution;
14 wherein upon encountering the commit instruction, the execution
15 mechanism is configured to successfully complete transactional execution of the

16 block of instructions preceding the commit instruction, wherein successfully
17 completing the transactional execution involves atomically committing changes
18 made during the transactional execution by:
19 treating store-marked cache lines as locked, thereby causing other
20 processes to wait to access the store-marked cache lines;
21 committing store buffer entries generated during transactional
22 execution to memory, wherein committing each store buffer entry
23 involves removing the store-mark from, and thereby unlocking, a
24 corresponding store-marked cache line;
25 clearing load-marks from cache lines; and
26 committing register file changes made during transactional
27 execution;
28 wherein changes made during the transactional execution are not
29 committed to the architectural state of the processor until the transactional
30 execution successfully completes.

1 11. (Previously Presented) The computer system of claim 10, wherein
2 while successfully completing transactional execution, the execution mechanism
3 is configured to:
4 resume normal non-transactional execution.

1 12. (Cancelled)

1 13. (Original) The computer system of claim 10, wherein if an
2 interfering data access from another process is encountered during the
3 transactional execution and prior to encountering the commit instruction, the
4 execution mechanism is configured to:
5 discard changes made during the transactional execution; and to

6 attempt to re-execute the block of instructions.

1 14. (Previously Presented) The computer system of claim 10, wherein
2 if a variation of the commit instruction is encountered, the execution mechanism
3 is configured to:

4 commence transactional execution of the block of instructions following
5 the commit instruction.

1 15. (Original) The computer system of claim 10, wherein the computer
2 system is configured to allow potentially interfering data accesses from other
3 processes to proceed during the transactional execution of the block of
4 instructions.

1 16. (Original) The computer system of claim 10, wherein the block of
2 instructions to be executed transactionally comprises a critical section.

1 17. (Original) The computer system of claim 10, wherein the commit
2 instruction is a native machine code instruction of the processor.

1 18. (Original) The computer system of claim 10, wherein the commit
2 instruction is defined in a platform-independent programming language.

1 19. (Currently Amended) A computer-readable storage medium
2 storing instructions that when executed by a computer cause the computer to
3 perform a method for executing a commit instruction to facilitate transactional
4 execution, comprising:

5 executing a block of instructions transactionally, wherein executing the
6 block of instructions transactionally involves placing load-marks on cache lines

7 from which data is loaded, placing store-marks on cache lines to which data is
8 stored, and placing transactional stores in a store buffer in the processor during
9 the transaction, wherein the transactional stores are gated and not committed to
10 memory from the store buffer during the transaction; and placing store-marks on
11 cache lines to which data is stored;

12 encountering the commit instruction during execution of a program,
13 wherein the commit instruction marks the end of a block of instructions to be
14 executed transactionally; and

15 upon encountering the commit instruction, successfully completing
16 transactional execution of the block of instructions preceding the commit
17 instruction, wherein successfully completing the transactional execution involves
18 atomically committing changes made during the transactional execution by:

19 treating store-marked cache lines as locked, thereby causing other
20 processes to wait to access the store-marked cache lines;

21 committing store buffer entries generated during transactional
22 execution to memory, wherein committing each store buffer entry
23 involves removing the store-mark from, and thereby unlocking, a
24 corresponding store-marked cache line;

25 clearing load-marks from cache lines; and

26 committing register file changes made during transactional
27 execution;

28 wherein changes made during the transactional execution are not
29 committed to the architectural state of the processor until the transactional
30 execution successfully completes.

1 20. (Previously Presented) The computer-readable storage medium of
2 claim 19, wherein successfully completing transactional execution involves:
3 resuming normal non-transactional execution.